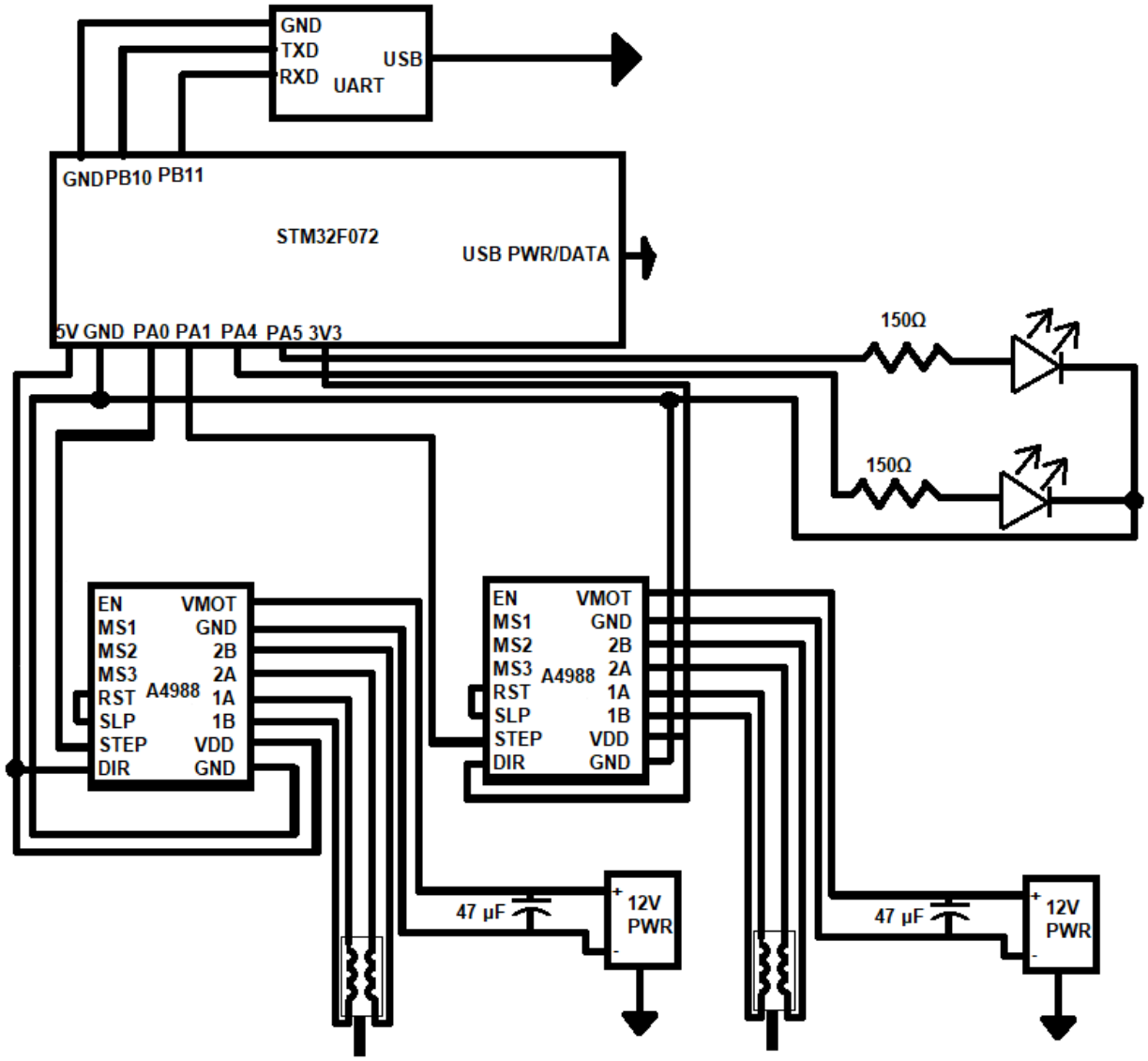
Create music with Stepper motors

Team: Phelan Hobbs and Seth Jackson



Link to the project files here: https://github.com/kieblade/Musical-Stepper-Motors.

The point of this project is to build an instrument using 2 motors that spin at different speeds to make different notes. The way we did this is by reading a .txt file through UART(Universal Asynchronous Receiver Transmitter) into the microcontroller. The microcontroller will then use the instructions from this file to command 2 stepper motors to rotate at various speeds to create different pitches.

When creating the text file music instructions, we used two lines for each note. The first line featured a capitalized letter representing the note, optionally a # to make a note sharp, and a number up to 8 representing the octave. The second line features a number representing how long the note would play. Alternatively, if a rest is requested, the string “off” would be featured in place of the note, again with the duration of this on the next line. For example, a song that plays a low A for 100 ticks, high B for 375 ticks, a middle C# for 175 ticks, and rests for 1000 ticks would be written like this:

A1

100

B7

375

C#5

175

off

1000

```
165   // yellow to PB10 orange to PB11 black to gnd.
166
167   GPIOB->MODER |= (1 << 21) | (1 << 23);
168   GPIOB->AFR[1] |= (1 << 14) | (1 << 10);
169
170   USART3->BRR = HAL_RCC_GetHCLKFreq()/115200;
171   USART3->CR1 |= (1 << 5) | (1 << 3) | (1 << 2) | 1;
172
173   NVIC_EnableIRQ(USART3_4_IRQn);
174
175   NVIC_SetPriority(USART3_4_IRQn, 1);
```

```
722
723  void USART3_4_IRQHandler(void)
724  {
725    buffer[rear] = USART3->RDR;
726    rear++;
727    if (LRecived == 0)
728    {
729      LRecived = 1;
730    }
731  }
732
```

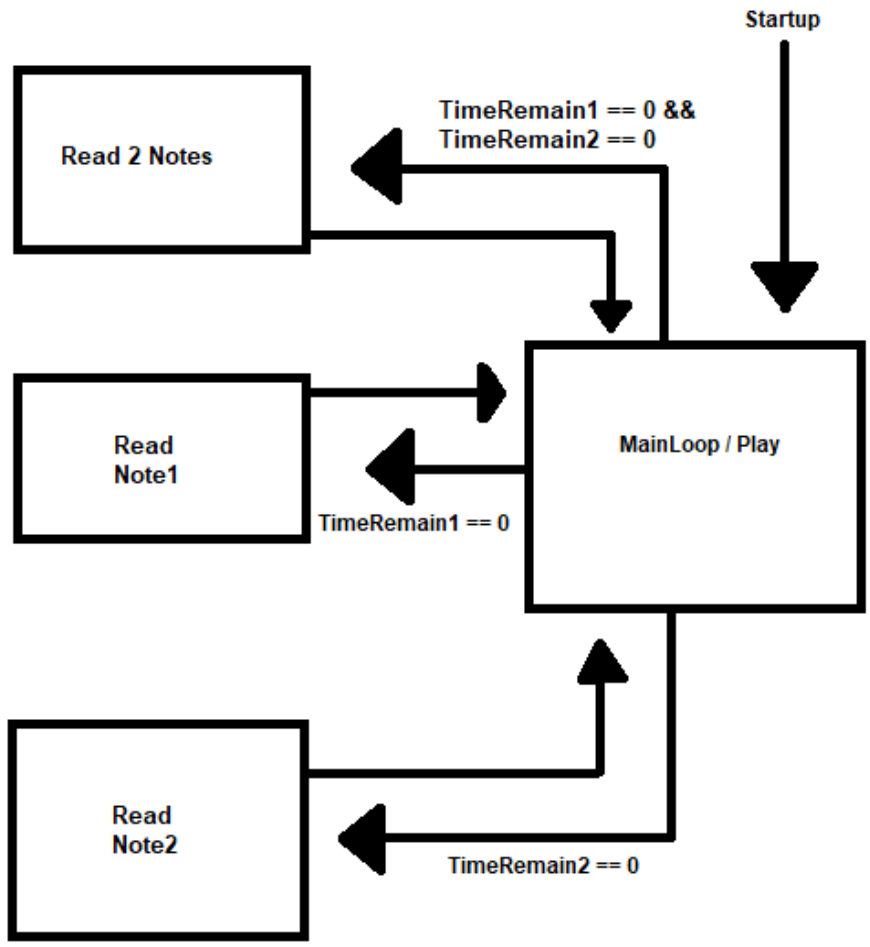
After defining the format for the musical instructions and setting up our UART cables for PB10 and PB11 for RXD and TXD respectively, we developed code to read from UART and

report it using the onboard lights. The entire file being sent across UART was placed into a large buffer character array for future use alongside a variable representing the index by utilizing the interrupt handler. Additionally, a char array “word” and an unsigned integer “timeremaining” were also defined prior to the main loop, with the latter being set to 0. When entering the main loop, if time remaining was equal to 0, it meant that the next two lines were to be read. The buffer was read and characters until “\r\n” were read and copied to the word char array. This was repeated for a secondary temporary char array which was then converted to the unsigned integer for time remaining. Otherwise, if the time remaining was not 0 at the start of the infinite loop, the code proceeded to either light up one of the onboard LEDs or dim it. This was accomplished by looking at the first character in the word, if it was “o”, the LED would dim, otherwise, it would brighten. We were able to send the text file over UART utilizing a saved PUTTY configuration that sent serial communication over COM3 by typing “Plink [Name of saved putty configuration] < [name of text file]” on the Windows command line.

The next step of this project was to verify that actual specific note was read correctly, we decided to use an analog output, specifically an LED, to emulate a specified tone visually. Doing this would allow us to fairly easily and quickly determine that tones were being read correctly and we decided to keep this, and the above mentioned on-board LEDs, in the final design as it allowed for quick bug fixing while also giving a visual element to the project as well. After configuring the DAC, we needed some way to convert “word” character array to an integer defined as “tone” before sending it to the DAC. We decided to split the character array into two different integer, offset and octave. The offset was determined by looking at the first char and assigned it based on its letter, with C being 0, D being 2, E being 4, F being 5, G being 7, A being 9, and B being 11. Additionally, we then checked the second character of the array, if, and only if, it was the hash symbol (ASCII character 35, “#”), it would increment this offset by 1. The octave was just just translated directly from the last full entry in the character array. We then converted these two smaller integers into a single unsigned integer ranging from 12 to 127 with the equation tone = 12 * (octave + 1) + offset + 127. The additional offset of 127 guaranteed that the LED was not too dim to be visible while lower notes were being played. Finally, if the note was “off”, the DAC would be set to output 0. On the hardware side of things, we decided to go with a low resistance resistor, specifically 150 ohms, to allow for a bright, undamaged light and tied the resistor and light in parallel with the PA4 pin on the discovery board. By analyzing the brightness, it was possible to ascertain the intended tone of the note, with low notes dimming the lights and high notes appearing much brighter.

To attach and control the 17HS16-2004s1 stepper motors, we used 16 pin A4988 motor drivers alongside two 12V wall power sources. The VDD and direction pins were attached to the 5V wire of our microcontroller alongside the grounds being connected. From there, we needed to adjust the A4988’s potentiometer such that 0.7V to 1.0V was measured across the potentiometer to ground so that neither the motor nor motor controller was damaged during operation. To be safely within the margins of both numbers, we decided to target as close to 0.85V as possible. As we did not need a handful of pins, the EN and MS1-3 pins were left unconnected and the reset and sleep pins were connected with each other. The 1A and 1B pins were connected across the stepper motor, this process was repeated for the 2A and 2B pins as well. The VMOT pin was connected to the positive output of a 12V power supply while the

power supply's ground was connected to the other ground of the microcontroller while a 47 microfarad capacitor separated these two lines. Finally, the step input of the microcontroller was connected with PA0 on our microcontroller. From there, it was relatively simple to adapt our previous code used to generate an analog signal to a specified note. We noticed that the frequencies of notes followed a specific pattern and that any note's frequency could be calculated based on the above offset and octave. In order to do this, we a base C as 16.35, C# as 17.32, D as 18.35, D# as 19.45, E as 20.6, F as 21.83, F# as 23.12, G as 24.5, G# as 25.96, A as 27.5, A# as 29.14, and B as 30.87. Additionally, a double was defined as frequency, which was calculated by freq = letterFrequency * (2 ^ octave) and turned to a period with period = 1 / frequency. Once the frequency is determined and the main loop was re-entered, a pulse was sent to PA0 at said frequency using the period variable.



To add an extra motor, a handful of helper functions were added namely TwoSetup, which is entered whenever two notes start being played at once, such as when a song starts, FirstExpire and SecondExpire, which sets up the notes for the respective motors, and calcFreq, which helps calculate the frequency. FirstExpire and SecondExpire are largely the same with the

exception of altering separate global variables and each read the next two lines from the character array. TwoSetup was, again, similar to these with the exception that it reads the next four lines rather than the next two. Finally calcFreq was just the previous frequency calculation in its own function to cut down on code reuse. This can be seen in the above flow chart. Every iteration through the main loop reduced the TimeRemain variables by one and were checked as soon as the loop started, going to the relevant function if required. Attaching the second motor and light was very similar to the first of the two components, with the light being attached to PA5 and the second motor controller being connected to PA1. As we only had a single 5V output on the board, the 3V3 was used instead with the potentiometer still set so that 0.85V was measured from it to ground. Each motor needed its own power source as they each individually required a minimum of 8V while our power supplies only provided 12V, with a 16V or greater power source, it is possible to run the two motors off of one outlet.